
# Updated NPSS S-function and Dynamic Analysis

Thomas Lavelle (LTA),
Jeffrey Csank (LCC), & Scott Jones (LTA)
NASA Glenn Research Center
Cleveland, OH, 44135

# Overview

- Introduction
  - New startup procedures
- NPSS Enhancements
  - Linearization
  - Changes to OD files
- Dynamic Systems Analysis and Application
  - Methodology
  - Mechanism for Analyzing Turbine Engine Dynamic Performance
  - Example model application

# NPSS MATLAB S-function

- NPSS S-function
  - NPSSSfunction.dll
    - NPSS V 1.6.5
    - MATALB S-function .dll support ended after R2010a

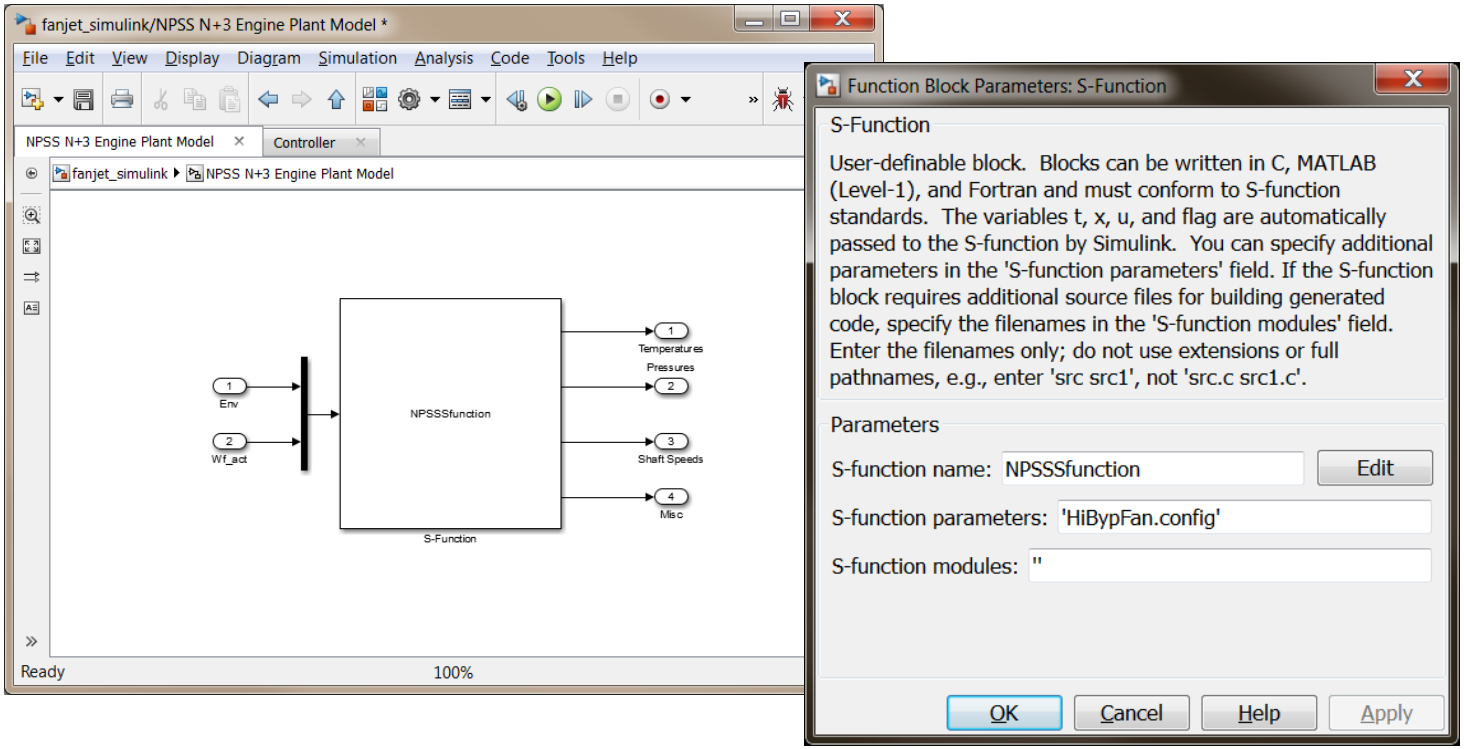
- Updated S-function (SWRI)
  - NPSS version 2.7.1
    - NPSSSfunction.mexw64
  - MATLAB®/Simulink version R2013b + 64-bit

# New MATLAB startup procedures

- Manual Process (SWRI)
  - Add NPSS bin to dos PATH
  - Open MATLAB from command prompt
  - Navigate to current MATLAB folder (model location)
  - Add location of MEX file to the MATLAB path
  - Open model
- Batch file (desktop)
  - Loads NPSS environment
  - Open MATLAB and run script to add NPSS path
- Install Paths
  - “setx” to permanently set NPSS environment for Windows
    - In Windows registry: “HKCU\Environment”
    - NPSS works immediately, can start MATLAB from anywhere

# Simulink Diagram

- Operation from Simulink remains unchanged.
  - NPSSSfunction and .config file are still used.
- NPSS crashing requires user to completely shutdown MATLAB and restart (time consuming).




# NPSS ENHANCEMENTS

# Config File

```
// Specify the time step
timeStep = 0.05;

// List of 'command line' type args that will be passed to the
// NPSS Command Interface when it is initialized.
// One of these args should be the name of the NPSS model input file.
commandLine = " -I . fanjet.run";

// Set up the ports to interface with Simulink.
SimulinkInPortMapper inPort0 {
   // specifying x names here will create a SimulinkInPortMapper with
   // an input vector of size x.
   // the Sfunction will take the values from the input vector and map them
   // into the NPSS variables corresponding to the given names.
   vars = { "altitude", "mach", "wfuel"}
}
…..
```

# Config File

```
SimulinkOutPortMapper outPort0 {
  // specifying x names here will create a SimulinkOutPortMapper with
  // an output vector of size x.
vars = { "Fan.Fl_I.Tt", "Splt.Fl_O1.Tt", "CmpH.Fl_I.Tt", "CmpH.Fl_O.Tt", "TrbH.Fl_I.Tt",
"TrbL.Fl_I.Tt", "TrbL.Fl_O.Tt" , "BFanOB.Fl_O.Tt" }
}

SimulinkOutPortMapper outPort1 {
  vars = { "Fan.Fl_I.Pt" ,"Splt.Fl_O1.Pt", "CmpH.Fl_I.Pt", "CmpH.Fl_O.Pt",
"TrbH.Fl_I.Pt", "TrbL.Fl_I.Pt", "TrbL.Fl_O.Pt", "CmpH.Fl_O.Ps"  , "BFanOB.Fl_O.Pt"}
}

….
```

# Added Features in Design File (desOD)

- Everything related to transient analysis included in this file
- Created by engine modeler for controls engineer
- Linear model generation
- Created guess values for model
  - Enables attempt at restarting model without restarting Simulink

# Linear Model

```
void create_linear_models(){
  // This function runs the NPSS model at
  // all the requested conditions for linear
  // model generation
  // sets the desired engine conditions and
  // then creates linear models output using
  // internal linear model generator
     .
     .
     .

}
```

# Linear Model

```
// This function outputs the generated linear models in NPSS format
void linear_model_output(){
    .
    .
    .

}
```

# Linear Model

```
// This function outputs the generated linear models in Matlab //format
void linear_model_matlab_output(){
…
        Very long function
…
}
```

# Guesses

```
real results[];
string indep[];
real alt[] = { 0, 5000, 10000, 20000, 40000};
real MN[][]={ { 0., .1, .3 },{ 0., .1, .3, .5 },{ 0., .1, .3, .5 },{ .4, .5, .6 },{ .7, .8, .85 }};
real PC[]={ 50, 45, 40, 35, 30 };

real wf[];
//---------------------------------------------------
// Run a throttle hook using the controller
//---------------------------------------------------
//---------------------------------------------------
// Run a throttle hook using the controller
//---------------------------------------------------

Independent FuelControl {
                varName = "BrnPri.Wfuel";
}

Dependent ControlErr {
                eq_lhs = "Fan.S_map.NcMap";
                eq_rhs = "CONTROL.PLACS*2./100.";
}


solver.addIndependent( "FuelControl" );
solver.addDependent( "ControlErr" );

indep=solver.list( "Independent" );
//real results[];
```


# Guesses

```
real results[];
string indep[];
real alt[] = { 0, 5000, 10000, 20000, 40000};
real MN[][]={ { 0., .1, .3 },{ 0., .1, .3, .5 },{ 0., .1, .3, .5 },{ .4, .5, .6 },{ .7, .8, .85 }};
real PC[]={ 50, 45, 40, 35, 30 };

real wf[];
//---------------------------------------------------
// Run a throttle hook using the controller
//---------------------------------------------------
//---------------------------------------------------
// Run a throttle hook using the controller
//---------------------------------------------------

Independent FuelControl {
                varName = "BrnPri.Wfuel";
}

Dependent ControlErr {
                eq_lhs = "Fan.S_map.NcMap";
                eq_rhs = "CONTROL.PLACS*2./100.";
}


solver.addIndependent( "FuelControl" );
solver.addDependent( "ControlErr" );

indep=solver.list( "Independent" );
//real results[];
```

# Guesses

```
void saveGuesses(){
   // This function runs through the envelop provided and saves
   //  of the solver independents for use to try to restore model


}
```

# Guesses

```
void setGuesses(){
    // Function is called to load in converged independents from
    // closest point (saveGuesses) to start solver or help model
    // recover

}
```

# Prep Model to Run

```
saveGuesses();

solverSequence.remove( "CONTROL" );
solver.removeIndependent( "FuelControl" );
solver.removeDependent( "ControlErr" );
presolverSequence = { "Guesses" }
postsolverSequence = {"ncpView"}

firstTime = 0;
altitude =30000.;
mach = .85;
wfuel = .55;
solver.maxIterations = 200;

run();
ncpView.display();


setOption( "solutionMode", "TRANSIENT" );

firstTime = 3;
```

# Run Before Model Execution At Every Step

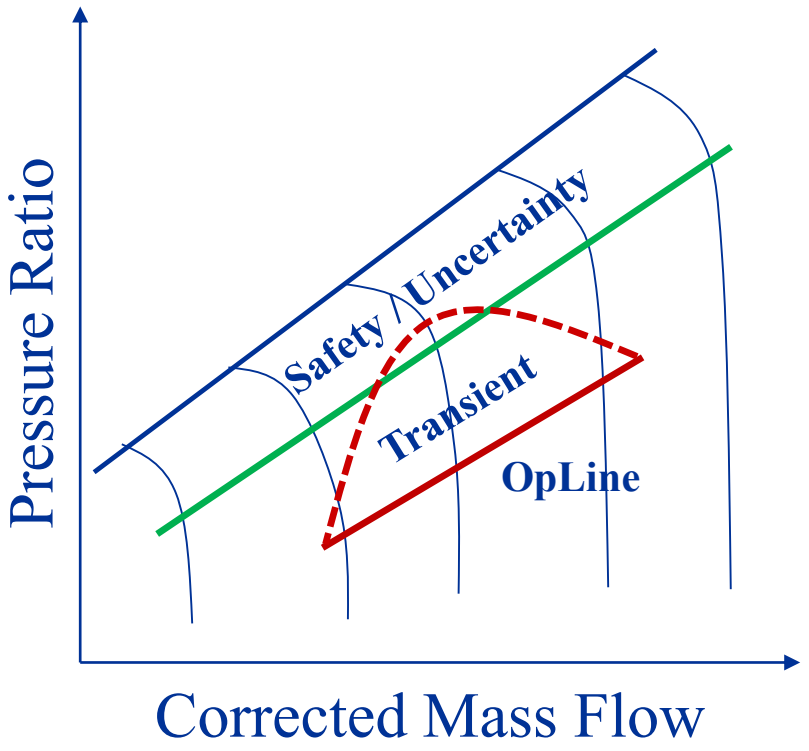
```
void Guesses() {
                .
                .
                .
                }
                if ( firstTime == 3 ){
                   // Check to be sure that input conditions from MATLAB match where NPSS is siting (Starting point)
                        .
                        .
                         .
                }
                if (fail==99){
                   //solver not coverged from previous pass, try to use guesses to recover
                    .
                    .
                    .
                }
}
```


# DYNAMIC SYSTEMS ANALYSIS

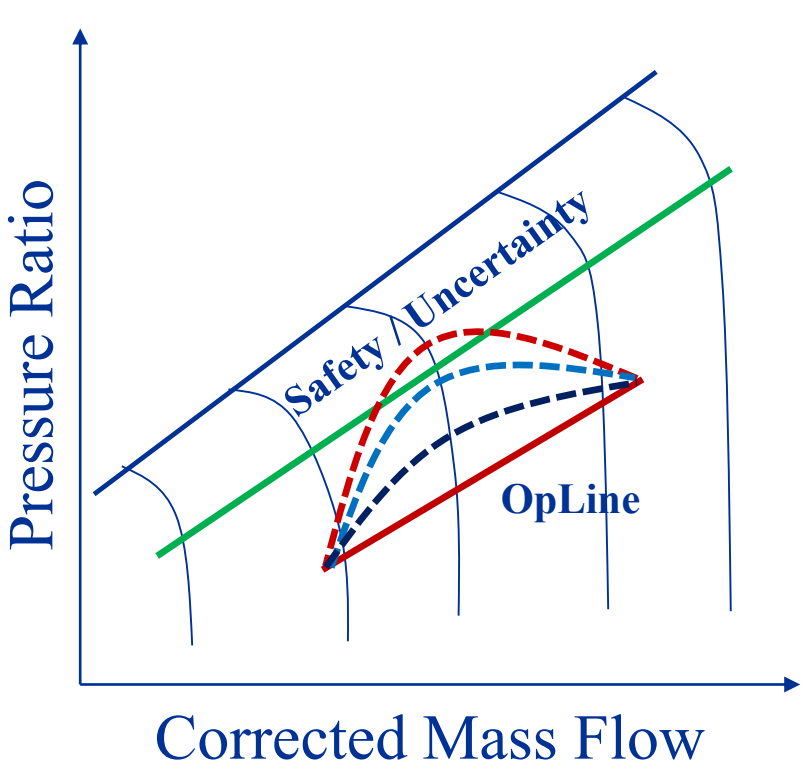
# Dynamic Operation

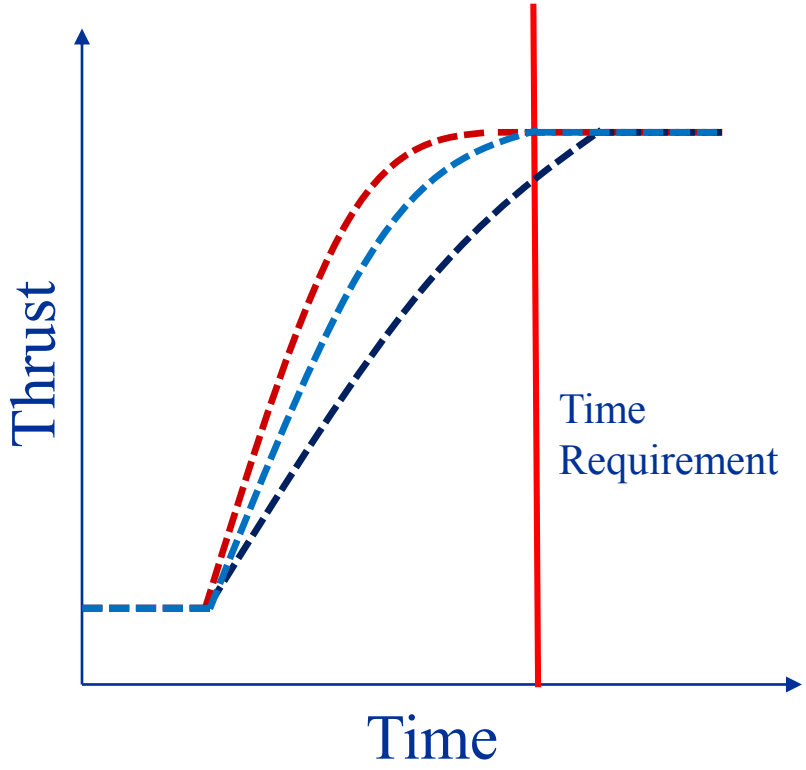
- OpLine (Steady-state)
  - Where engine operates in steady-state.
  - Operate closer to surge = increased efficiency.
  - Operate farther from surge = decreased efficiency.

- Why not operate closer to surge for better efficiency?
  - Safety (Uncertainty)
  - Transient Operation



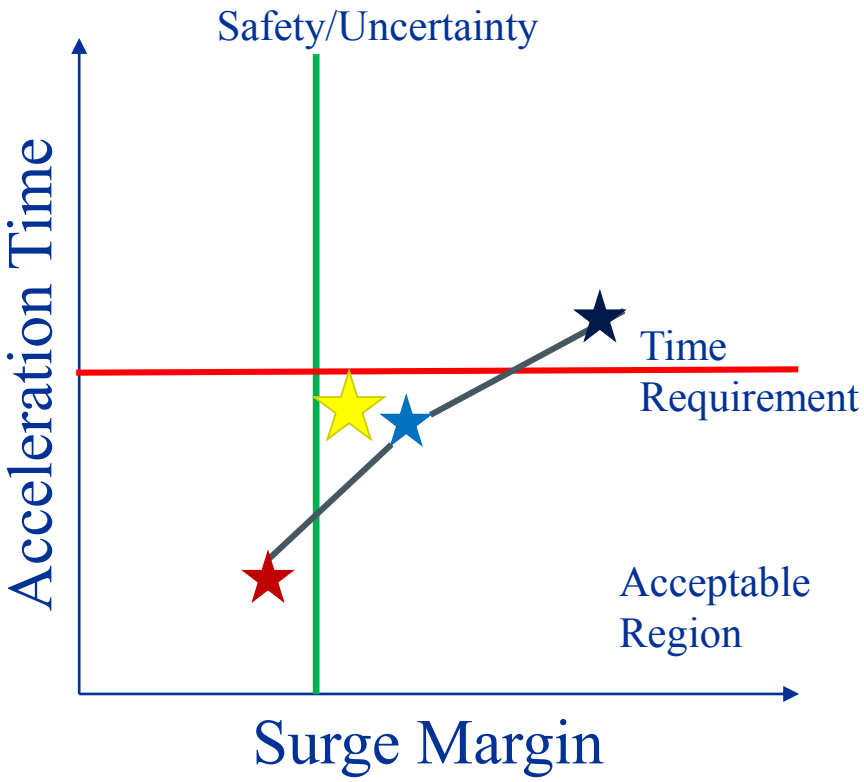
# Dynamic Operation




- This type of closed-loop simulation data contains:
  - worst surge margin (minimum), and
  - dynamic response time.

# Mechanism for Analyzing Turbine Engine Dynamic Performance

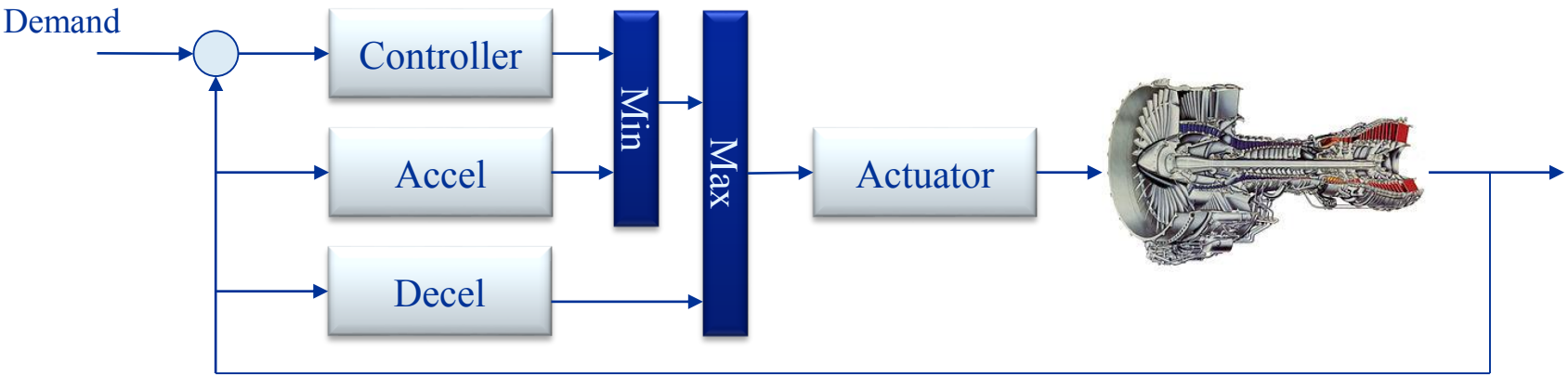



- What is the ideal design?
  - No benefit to excessively fast acceleration time.
  - Designs with more margin (points further to the right) are less efficient.
  - Ideal point upper left hand corner of the Acceptable Region.
  - Plot can provide information to help guide system studies (analysis).

- Control system helps balance performance (time response) and operability (worst case surge margin).

# Tool for Turbine Engine Closed-loop Transient Analysis

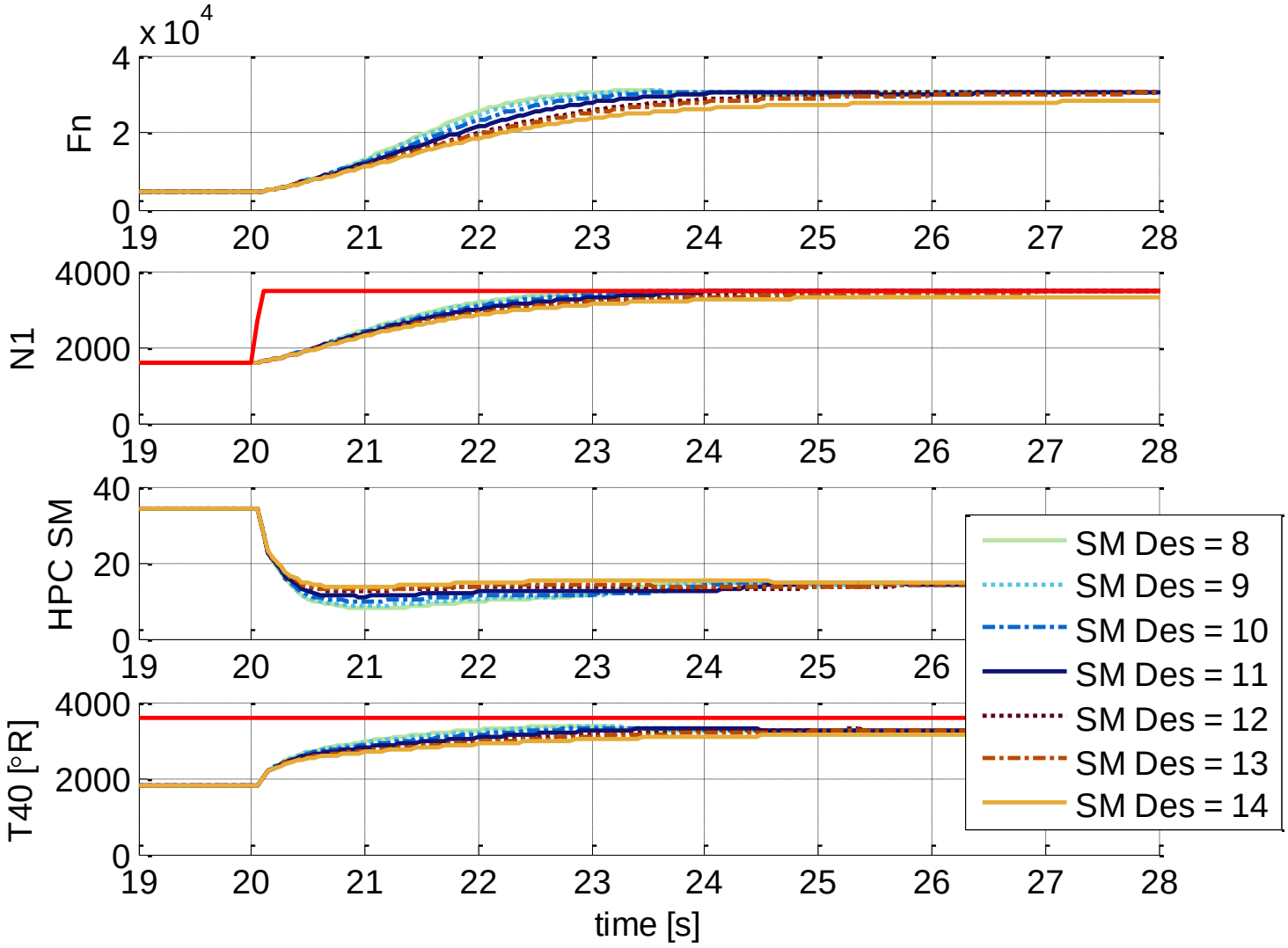
- Provide an estimate of the closed-loop transient performance of an engine model.
  - https://github.com/nasa/TTECTrA/releases
- Automatically designs a transient controller.
  - Set point function and controller
  - Acceleration controller (Surge Margin, T40)
  - Deceleration controller (Surge Margin, FAR)
- Integrated TTECTrA with NPSS S-function



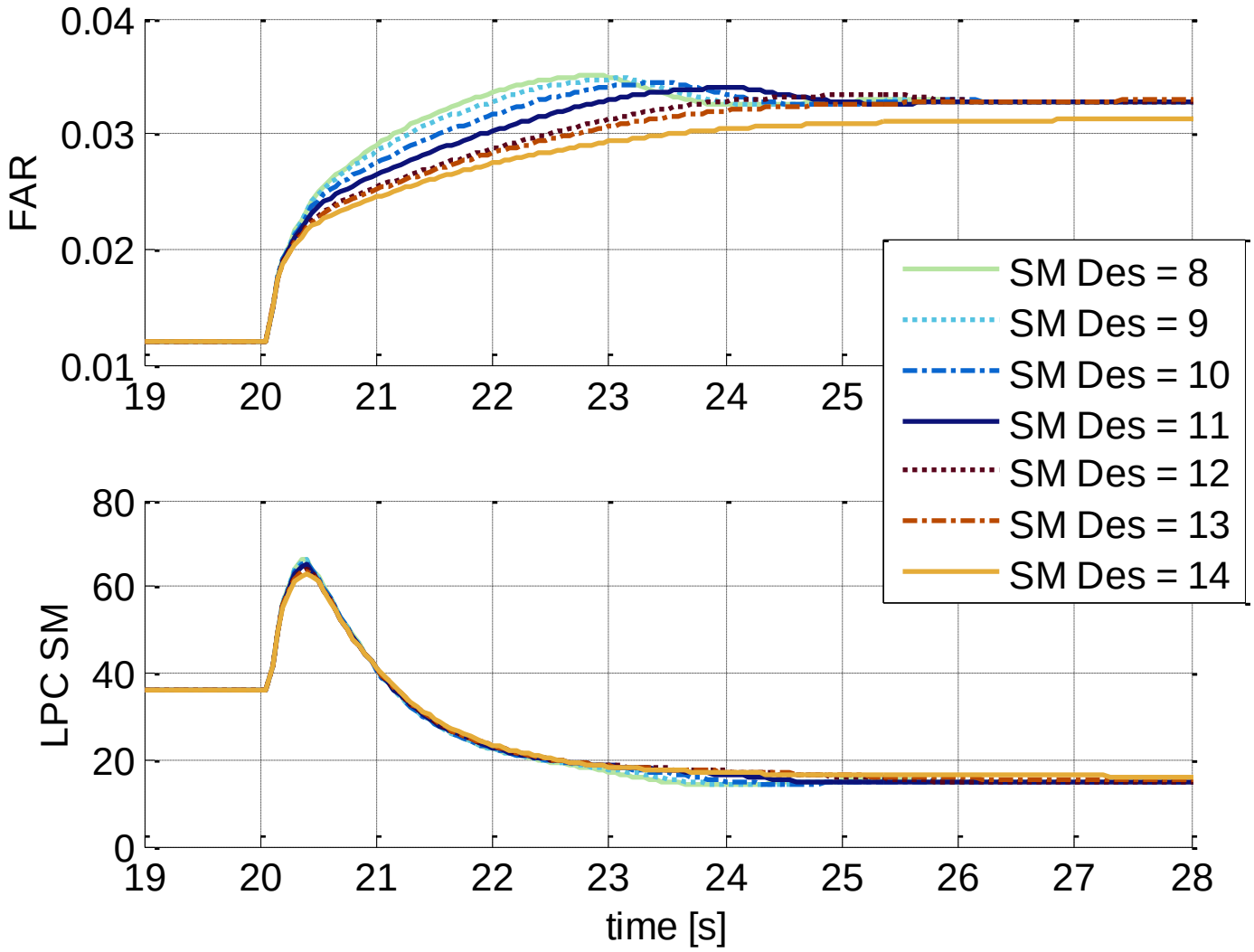
# Example Using NPSS S-function

- Advanced High Bypass Turbofan
- NPSS Demonstration Model

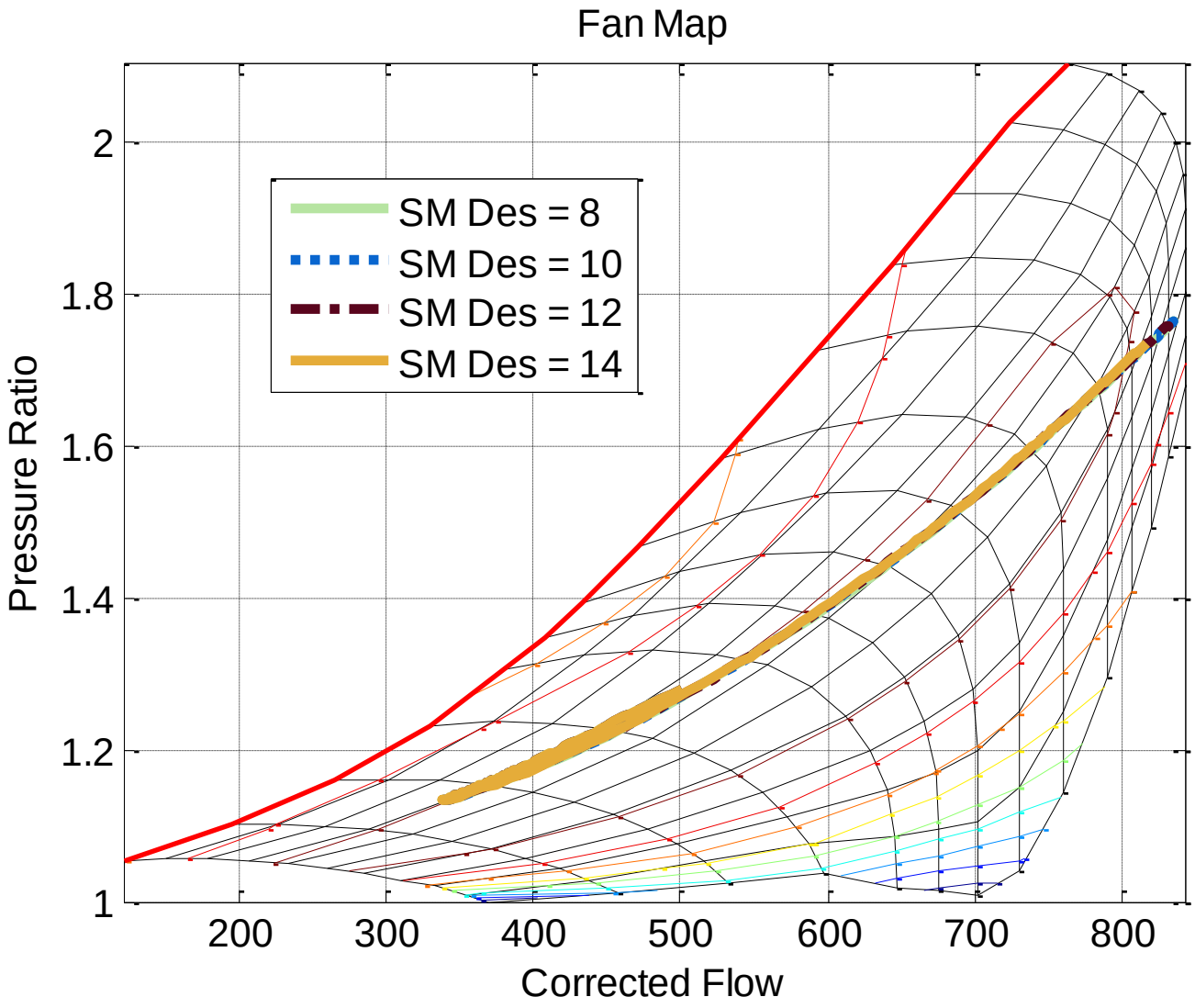
# Thrust Burst Profile
# Set Point and Acceleration Controllers



# Thrust Burst Profile
# Deceleration Controller



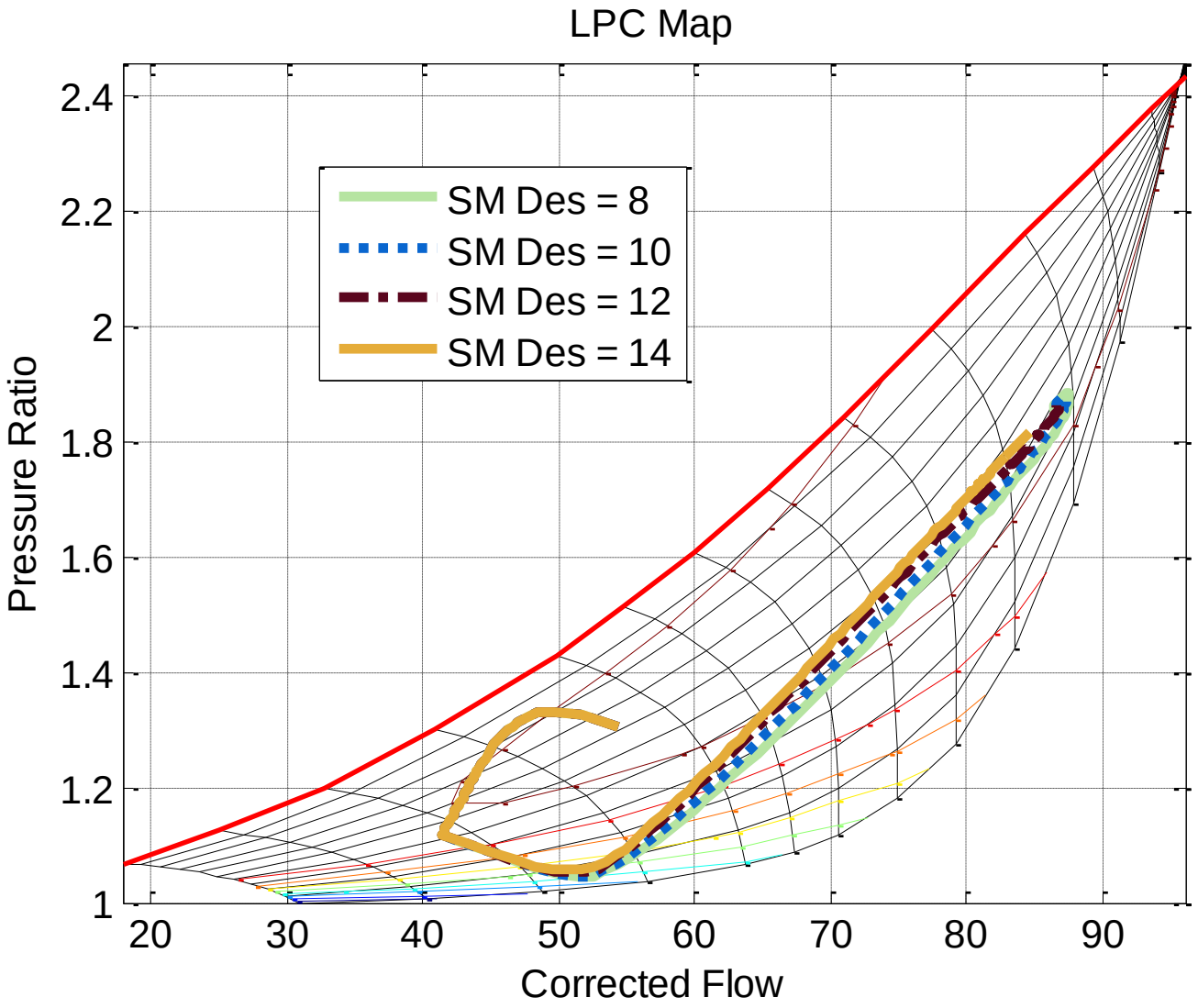
# Engine Fan Map



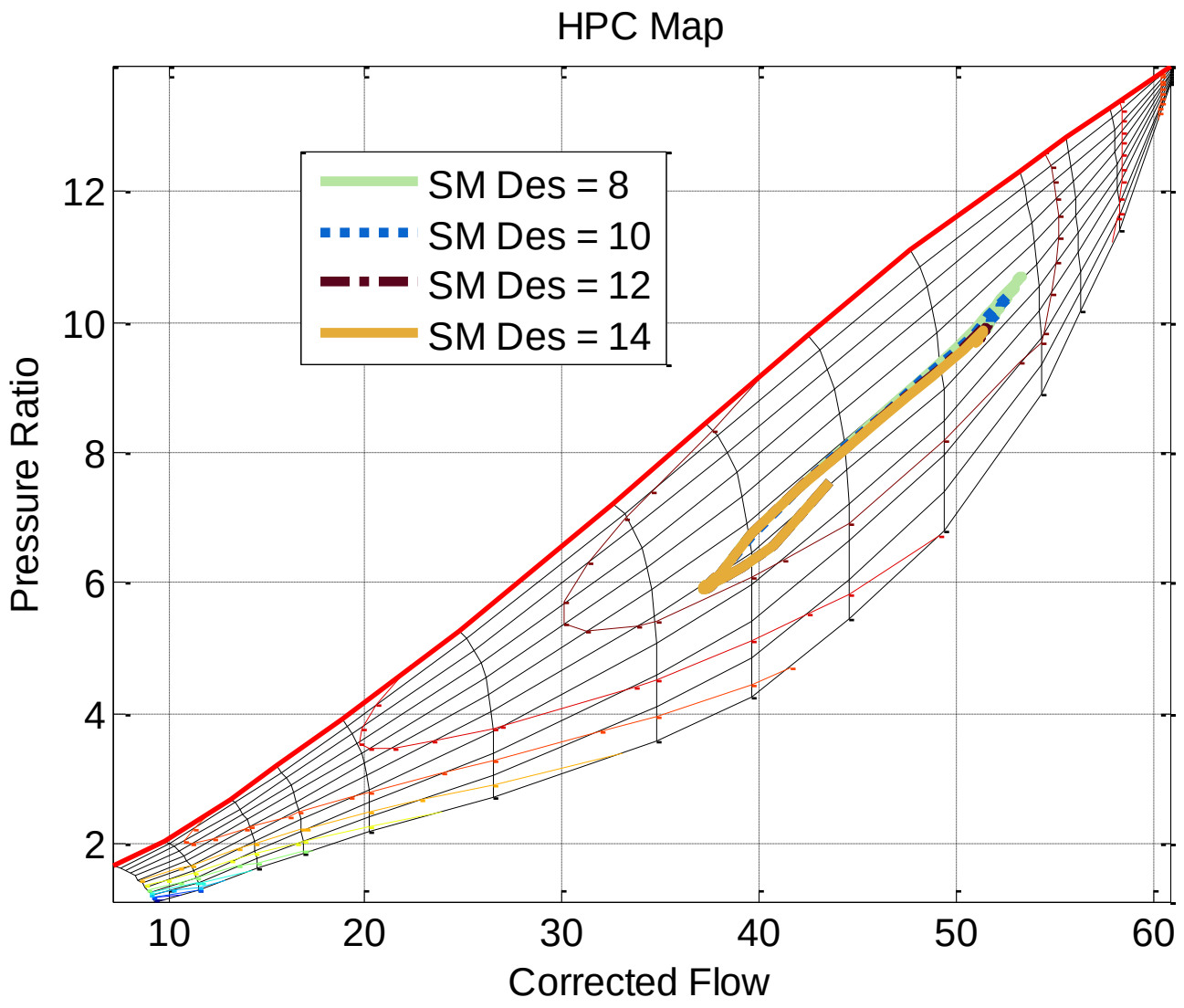

NPSS script added to save component maps in .m file for MATLAB to read and plot.

Added additional outputs from NPSS (Sfunction) to provide actual map data (Rline, Nc).

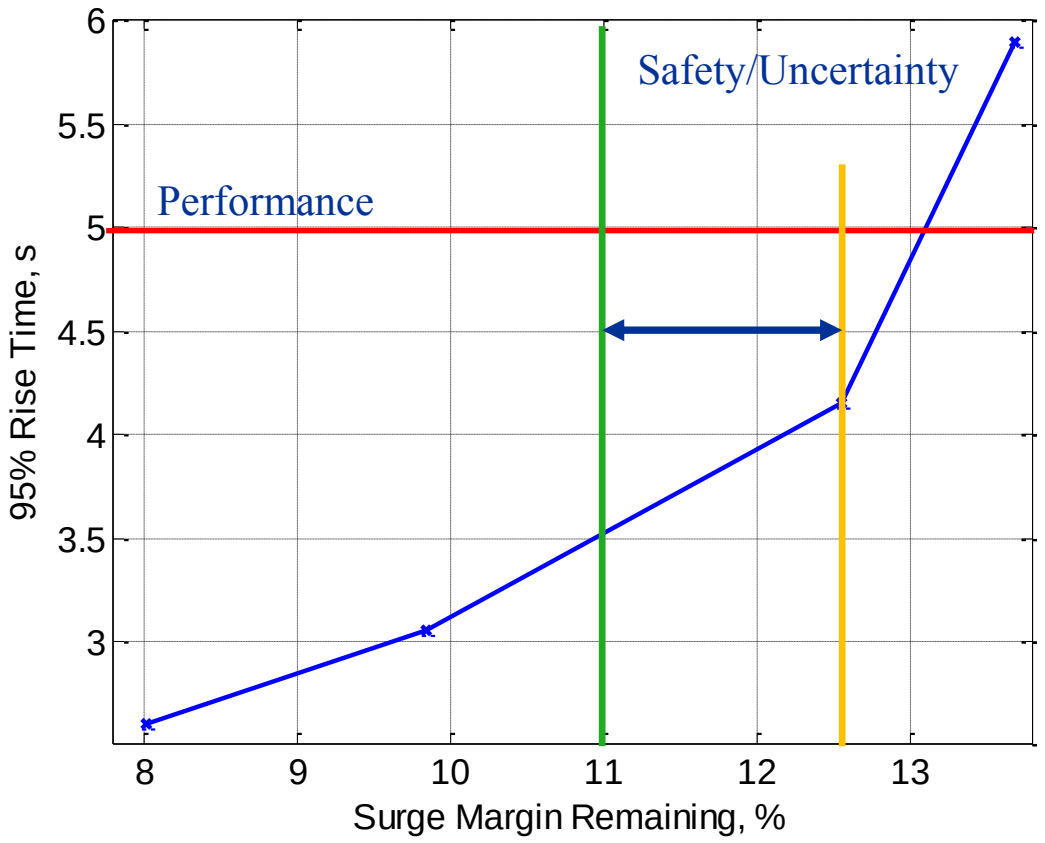
# Low Pressure Compressor Map



# High Pressure Compressor Map



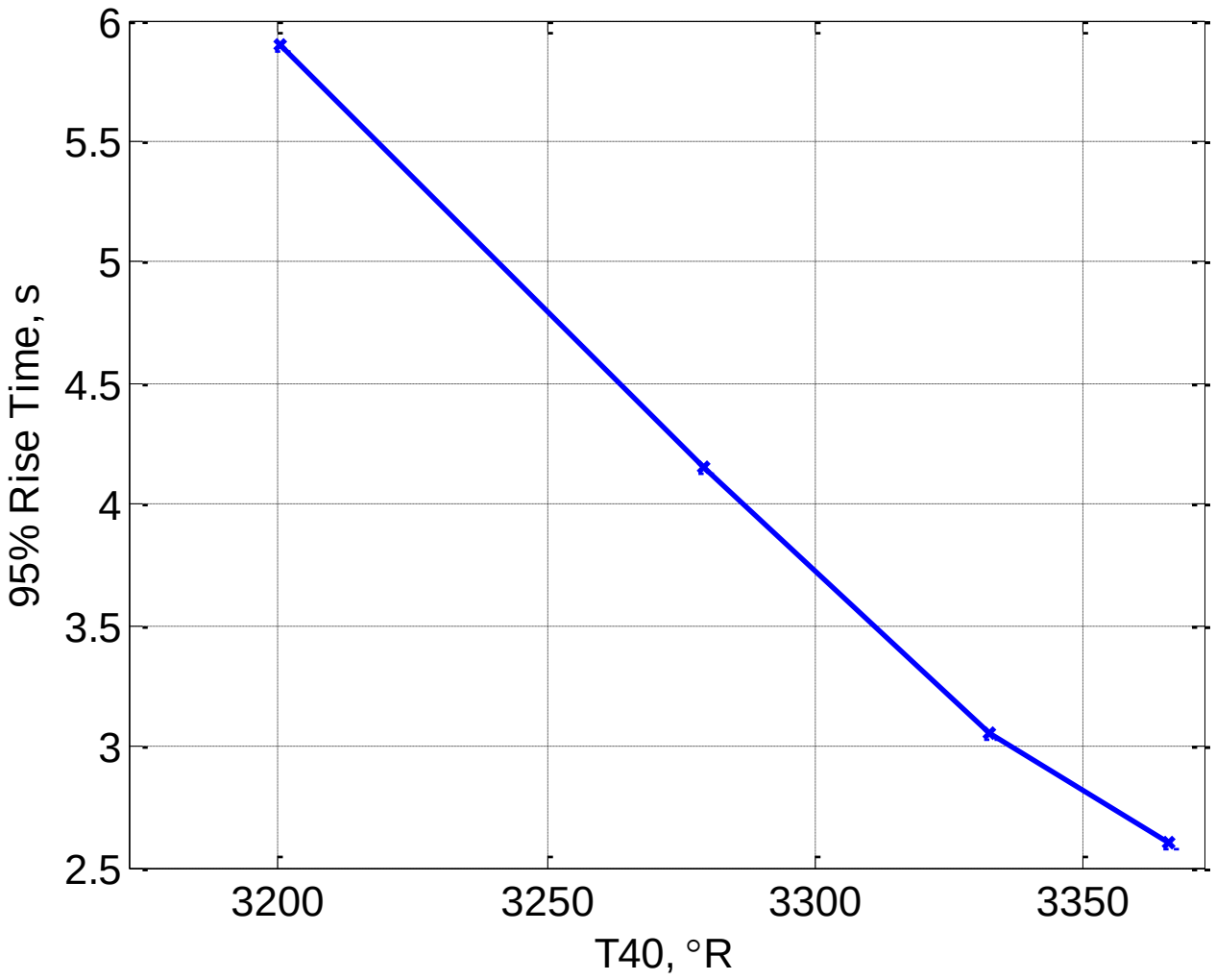
# Mechanism for Analyzing Turbine Engine Dynamic Performance




With the safety, or uncertainty, stack at 11%, the current design **may** have additional margin that could be traded for better performance (weight reduction, efficiency, etc.)

Modify surge margin constraint and allow a redesign.

# T40 vs Rise Time



# Summary

- NPSS Compiled using 64 bit compiler (SWRI).
- NASA tested and added functionality for control design users (linearization, MATLAB compatibility).
- Updated TTECTrA for newer versions of MATLAB and to better integrate with NPSS.
- Provided example of the type of design information available through the dynamic systems analysis process.
  - Relationship between surge margin and acceleration time.
  - Allow conservative margin to be traded for increased efficiency.